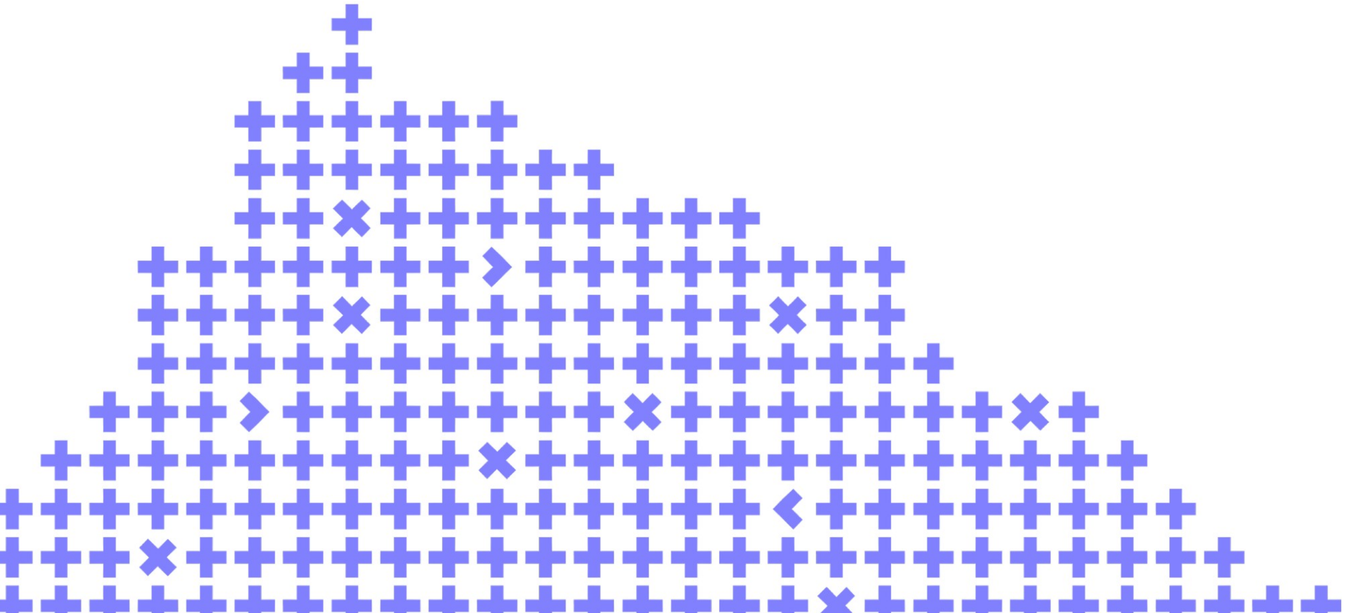
Transactional Queues in PostgreSQL

Igor Loban



Title

- Body Level One
  - Body Level Two
    - Body Level Three
      - Body Level Four
        - Body Level Five

- 10 years in backend
- The last 6 years at Toloka.ai
- Lover of Java and PostgreSQL

About Toloka.ai

- Toloka is a data labeling service for ML
- 300 labels per second from real people
- 40 microservices
- Many clusters of PostgreSQL, MongoDB, ClickHouse, Redis
- SQS-like message brokers for async communication

About Toloka.ai

- Body Level One
  - Body Level Two
    - Body Level Three
      - Body Level Four
        - Body Level Five

- Toloka is a data labeling service for ML
- 300 labels per second from real people
- 40 microservices
- Many clusters of PostgreSQL, MongoDB, ClickHouse, Redis
- SQS-like message brokers for async communication

Goal

- Highlight the problem that forces to use transactional queues in PostgreSQL
- Give a foundation to solve that problem in your projects

- Body Level One
  - Body Level Two
    - Body Level Three
      - Body Level Four
        - Body Level Five

Content

01

Why should we use transactional queues?

Problem

- Distributed business transactions
- also known as consistent changes in different services

- Body Level One
  - Body Level Two
    - Body Level Three
      - Body Level Four
        - Body Level Five

Problem

- Distributed business transactions
- also known as consistent changes in different services
- Saga pattern is one of the solutions
- The main challenge is **an atomic change in a local DB and sending a message to a broker**

- Body Level One
  - Body Level Two
    - Body Level Three
      - Body Level Four
        - Body Level Five

Example 1

- Body Level One
  - Body Level Two
    - Body Level Three
      - Body Level Four
        - Body Level Five

```
@Transactional
public void updateOrder(Order order) {
  // transaction has been started
  orderRepo.save(order);

  writeToExternalQueue(order);

  // commit can fail!

}
```

Example 2

```
public void updateOrder(Order order) {
    saveOrderInTx(order); // DB transaction inside the method
    writeToExternalQueue(order); // can fail!
}
```

- Body Level One
  - Body Level Two
    - Body Level Three
      - Body Level Four
        - Body Level Five

Solution

- Background process for retries
- Boolean field “sent to queue” in an entity

- Body Level One
- Body Level Two
- Body Level Three
- Body Level Four
- Body Level Five

Solution

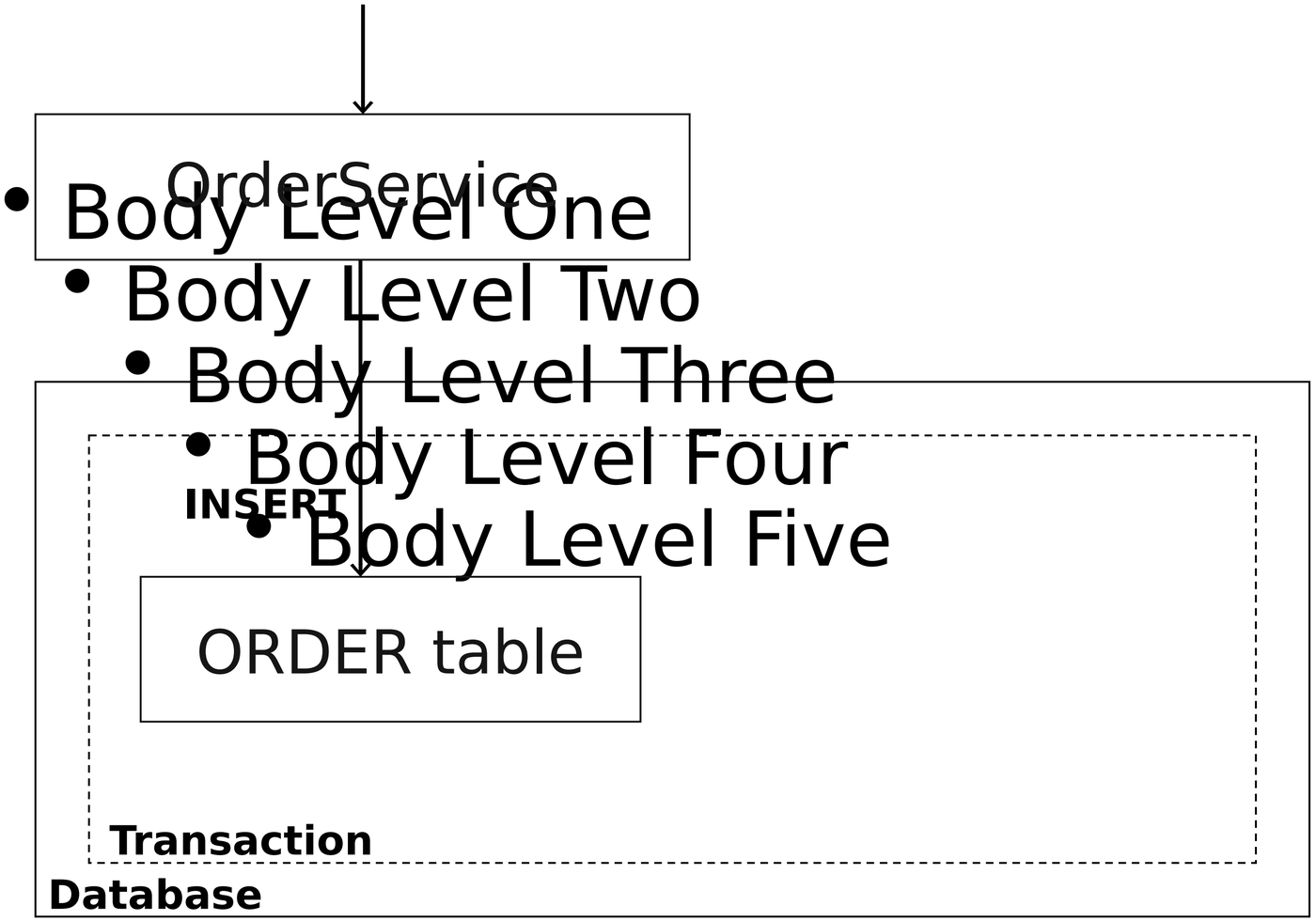
- Background process for retries
- Boolean field “sent to queue” in an entity
- Problem 1: potential performance degradation
- Problem 2: similar code in multiple places

- Body Level One
  - Body Level Two
    - Body Level Three
      - Body Level Four
        - Body Level Five

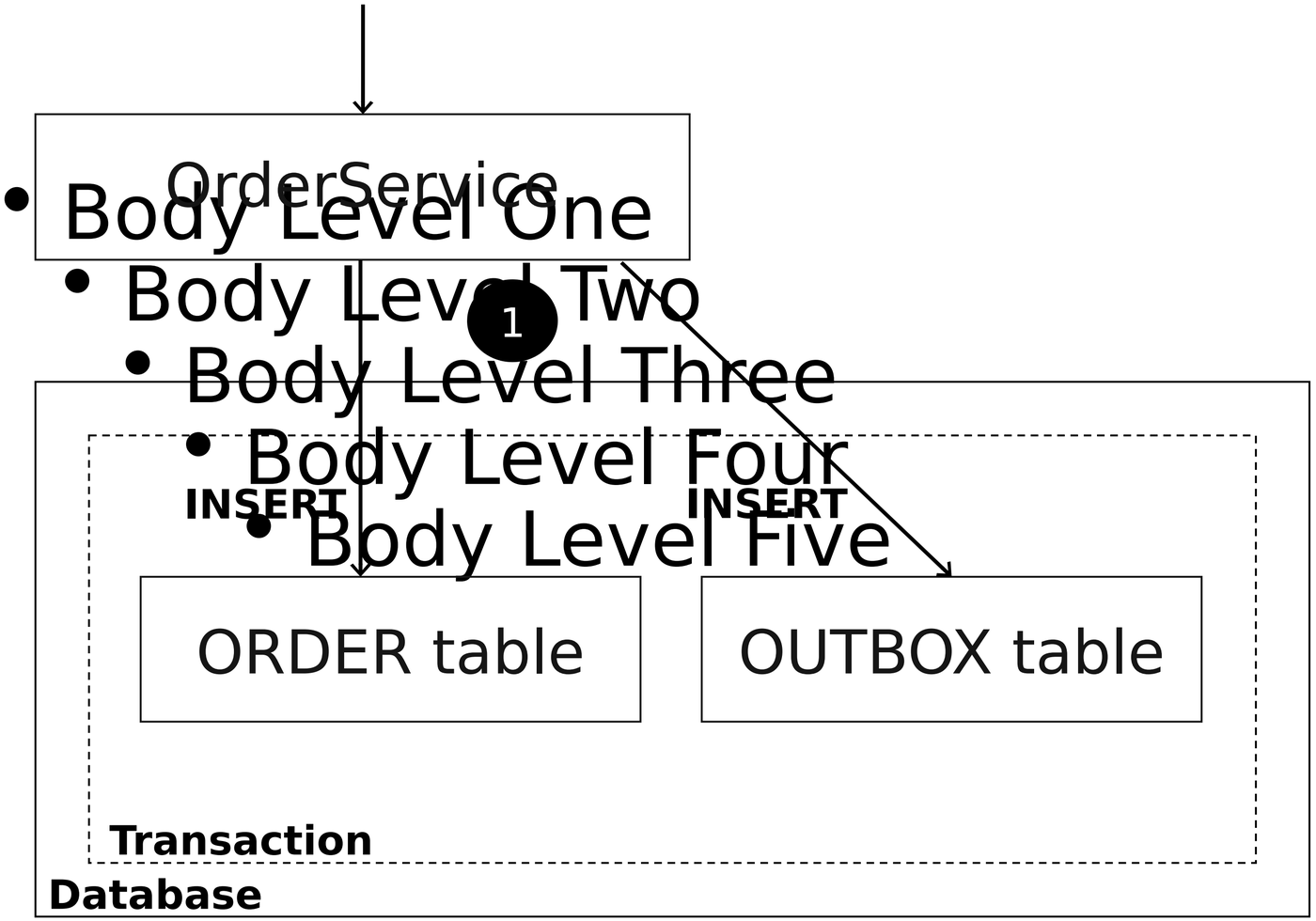
02

Transactional Outbox Design Pattern

Transactional Outbox

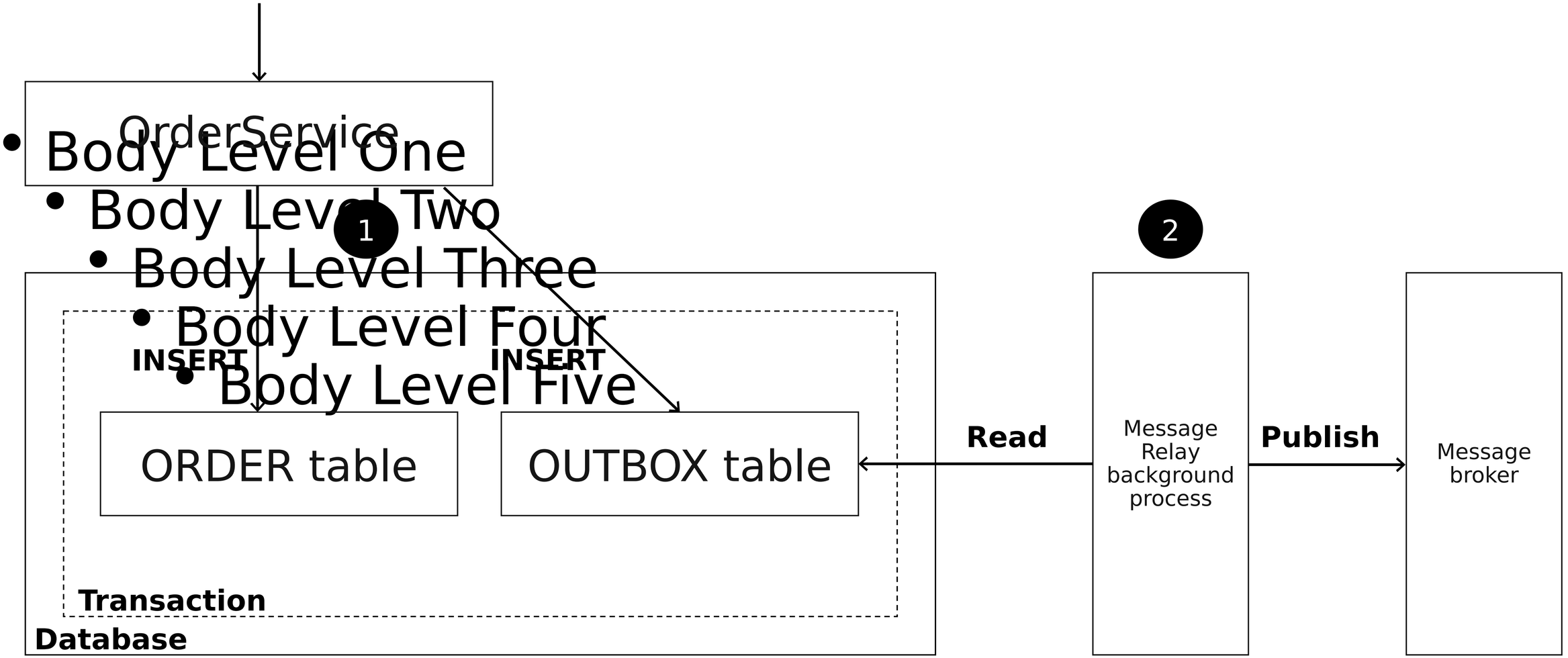


Transactional Outbox



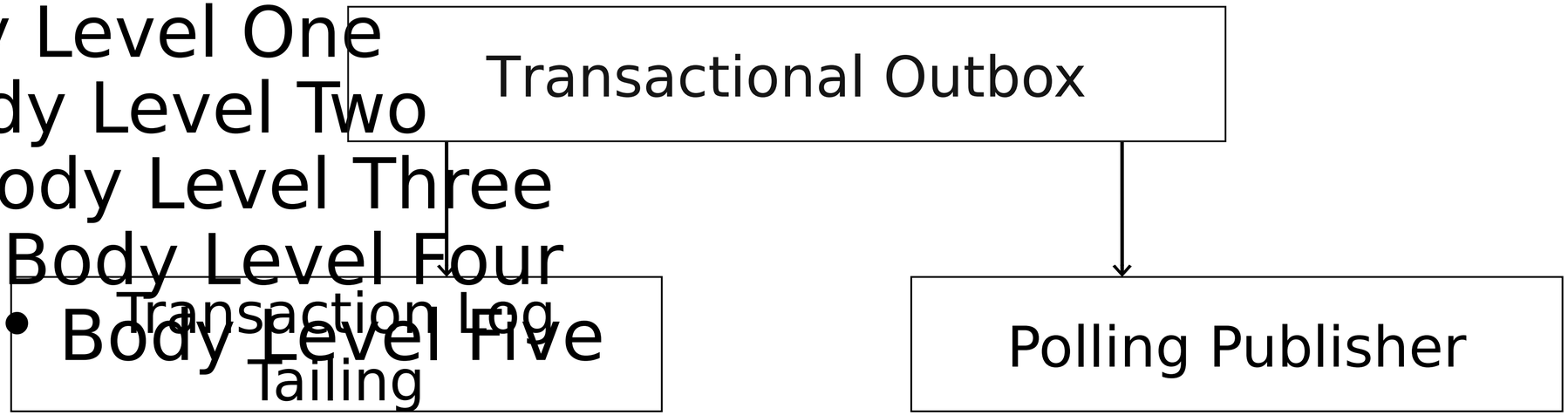
Transactional Outbox

- Body Level One
  - Body Level Two
    - Body Level Three
      - Body Level Four
        - Body Level Five



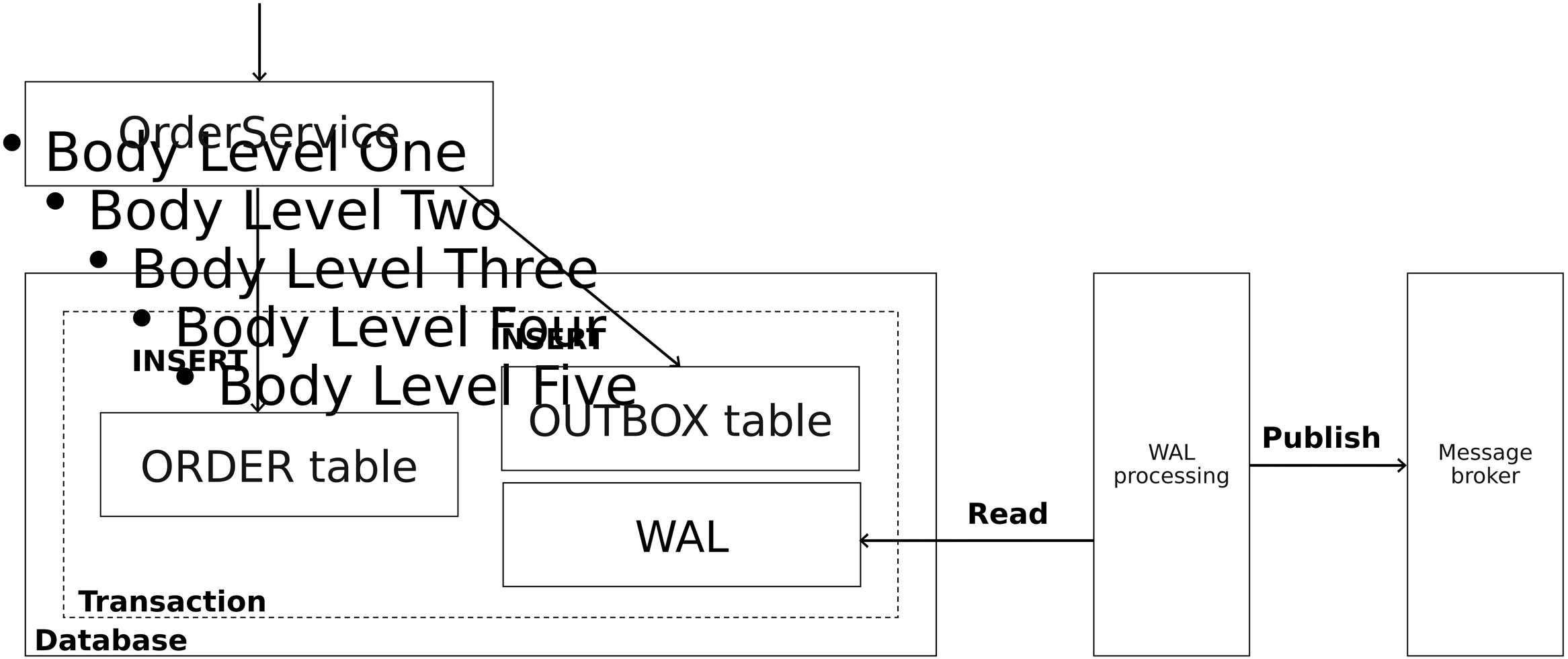
Transactional Outbox Subtypes

- Body Level One
  - Body Level Two
    - Body Level Three
      - Body Level Four
        - Body Level Five

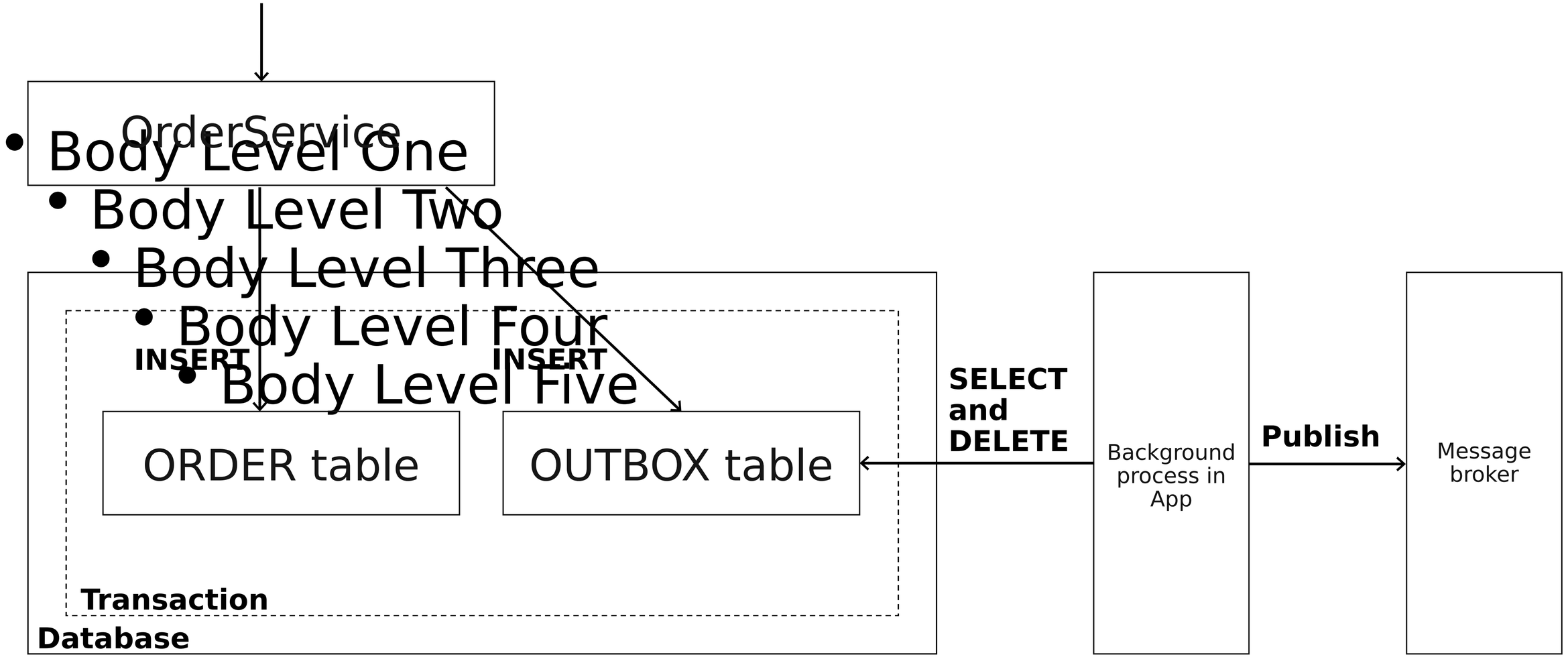


Transaction Log Tailing

- Body Level One
  - Body Level Two
    - Body Level Three
      - Body Level Four
        - Body Level Five



Polling Publisher



Long DB Transactions

- Business logic of your application
- Queries to an analytic replica (hot_standby_feedback)
- Index building
- pg_repack

- Body Level One
  - Body Level Two
    - Body Level Three
      - Body Level Four
        - Body Level Five

Long DB Transactions

- Business logic of your application
- Queries to an analytic replica (hot_standby_feedback)
- Index building
- pg_repack

- Body Level One
  - Body Level Two
    - Body Level Three
      - Body Level Four
        - Body Level Five

Long DB Transactions

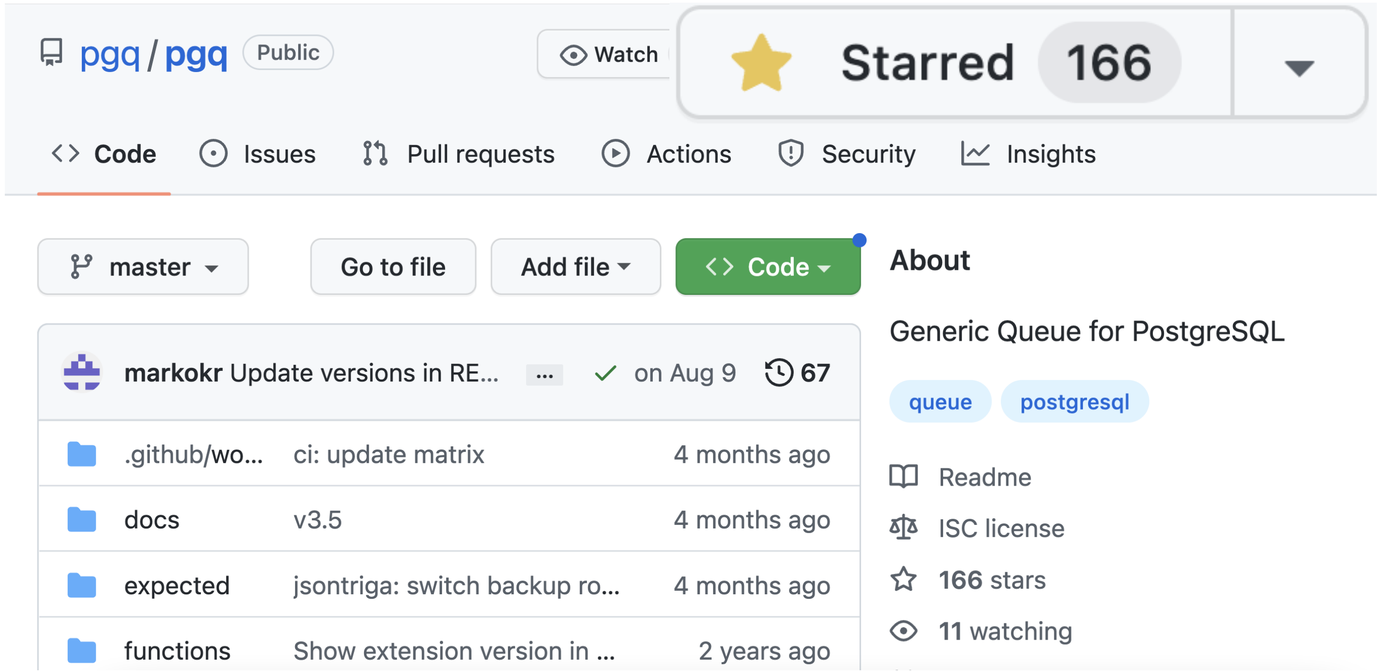
- Business logic of your application
- Queries to an analytic replica (hot_standby_feedback)
- Index building
- pg_repack

- Body Level One
  - Body Level Two
    - Body Level Three
      - Body Level Four
        - Body Level Five

03

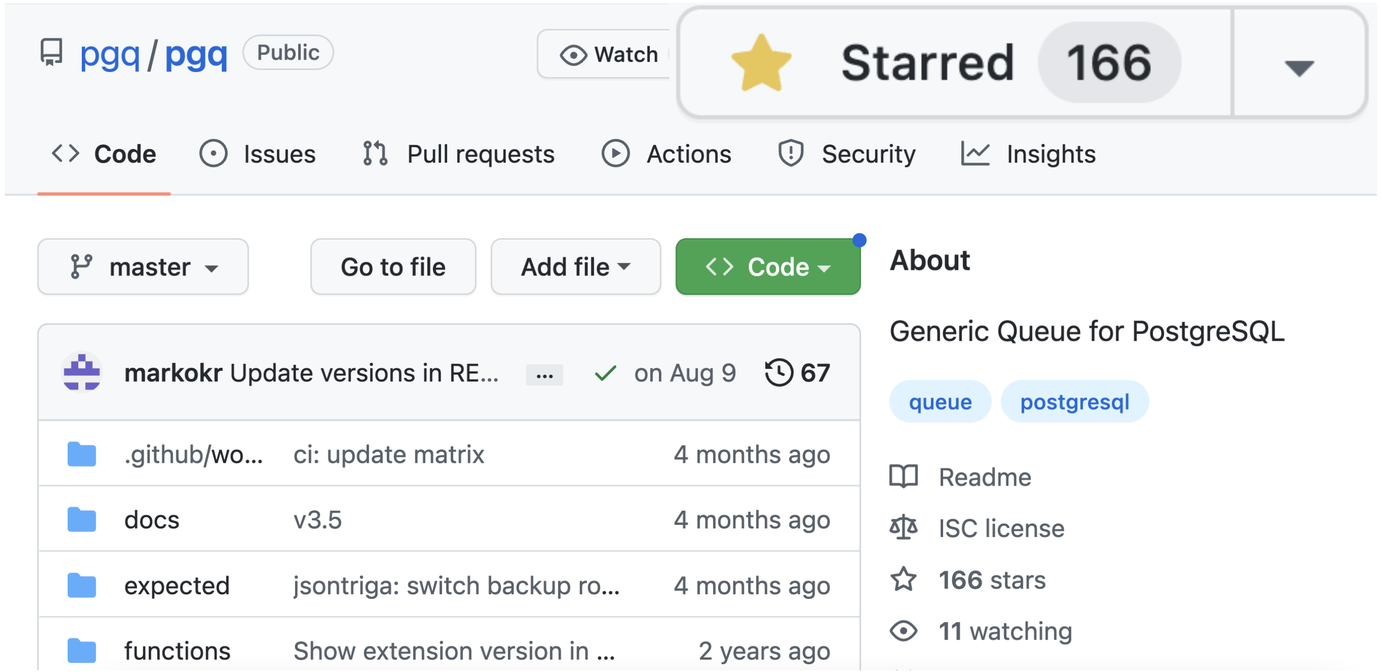
PgQ and Eventuate

PgQ



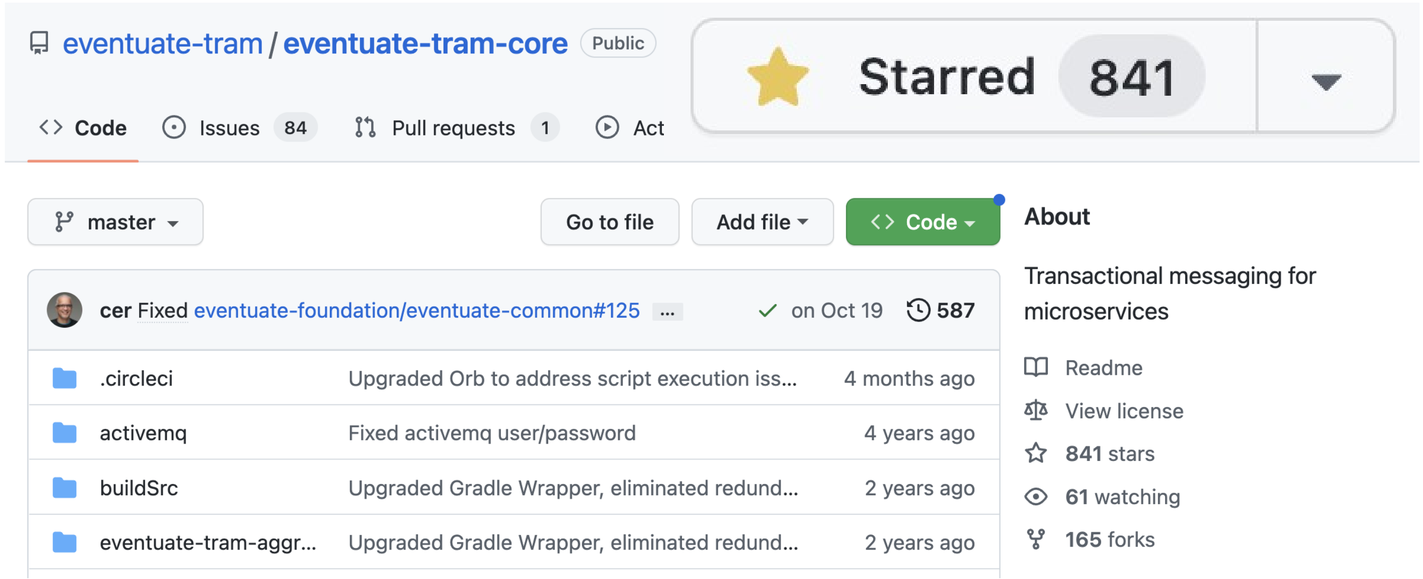
- Extension for PostgreSQL
- Full-fledged message broker based on PostgreSQL
- Requires process-daemon pgqd

PgQ



- Extension for PostgreSQL
- Full-fledged message broker based on PostgreSQL
- Requires process-daemon pgqd

Eventuate



- Framework for Transactional Outbox and Saga patterns
- PostgreSQL and MySQL – Transaction Log Tailing
- SQL Server – Polling Publisher

04

Recipe for Transactional Queues in PostgreSQL

Limitations

- Body Level One
  - Body Level Two
    - Body Level Three
      - Body Level Four
        - Body Level Five

At-least-once delivery

No ordering guarantee

Stress-test

- Disclaimer: it is not a benchmark
- 70 writers (1000 events per second)
- 5 readers (2000 events per second)
- Sync replica with 50 ms delay (recovery_min_apply_delay)
- Begin a long transaction and wait for 5 min

- Body Level One
  - Body Level Two
    - Body Level Three
      - Body Level Four
        - Body Level Five

Stress-test

- Disclaimer: it is not a benchmark
- 70 writers (1000 events per second)
- 5 readers (2000 events per second)
- Sync replica with 50 ms delay (recovery_min_apply_delay)
- Begin a long transaction and wait for 5 min

- Body Level One
  - Body Level Two
    - Body Level Three
      - Body Level Four
        - Body Level Five

Stress-test

- Disclaimer: it is not a benchmark
- 70 writers (1000 events per second)
- 5 readers (2000 events per second)
- Sync replica with 50 ms delay (recovery_min_apply_delay)
- Begin a long transaction and wait for 5 min

- Body Level One
  - Body Level Two
    - Body Level Three
      - Body Level Four
        - Body Level Five

Title FOR UPDATE

The OUTBOX table

- Body Level One
  - Body Level Two
    - Body Level Three
      - Body Level Four
        - Body Level Five

```
CREATE TABLE queue_buffer (
  id      BIGSERIAL PRIMARY KEY,
  payload TEXT
);
```

Title FOR UPDATE

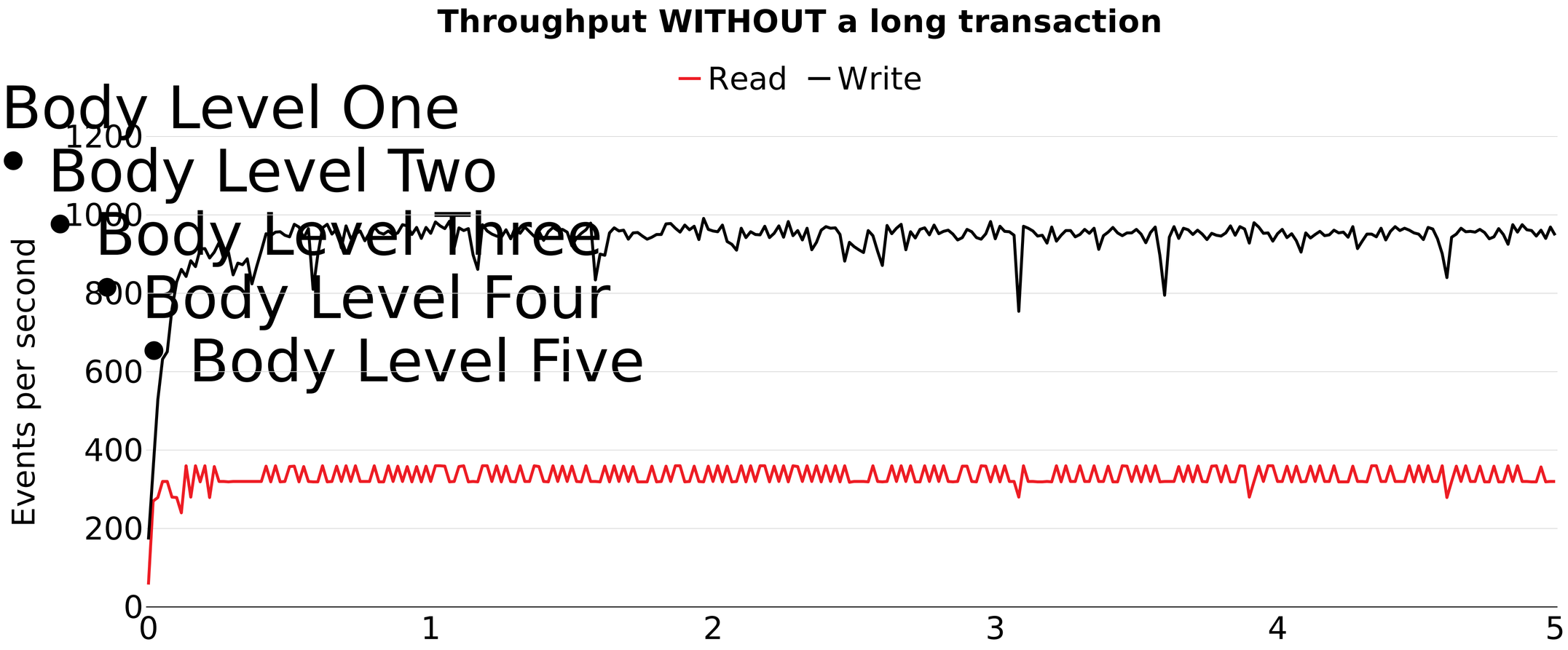
Readers lock rows, sleep, and delete them in a single transaction

- Body Level One
  - Body Level Two
    - Body Level Three
      - Body Level Four
        - Body Level Five

```
BEGIN;

SELECT id, payload FROM queue_buffer

ORDER BY id

FOR UPDATE

LIMIT ?;

…

DELETE FROM queue_buffer WHERE id IN (…);

COMMIT;
```

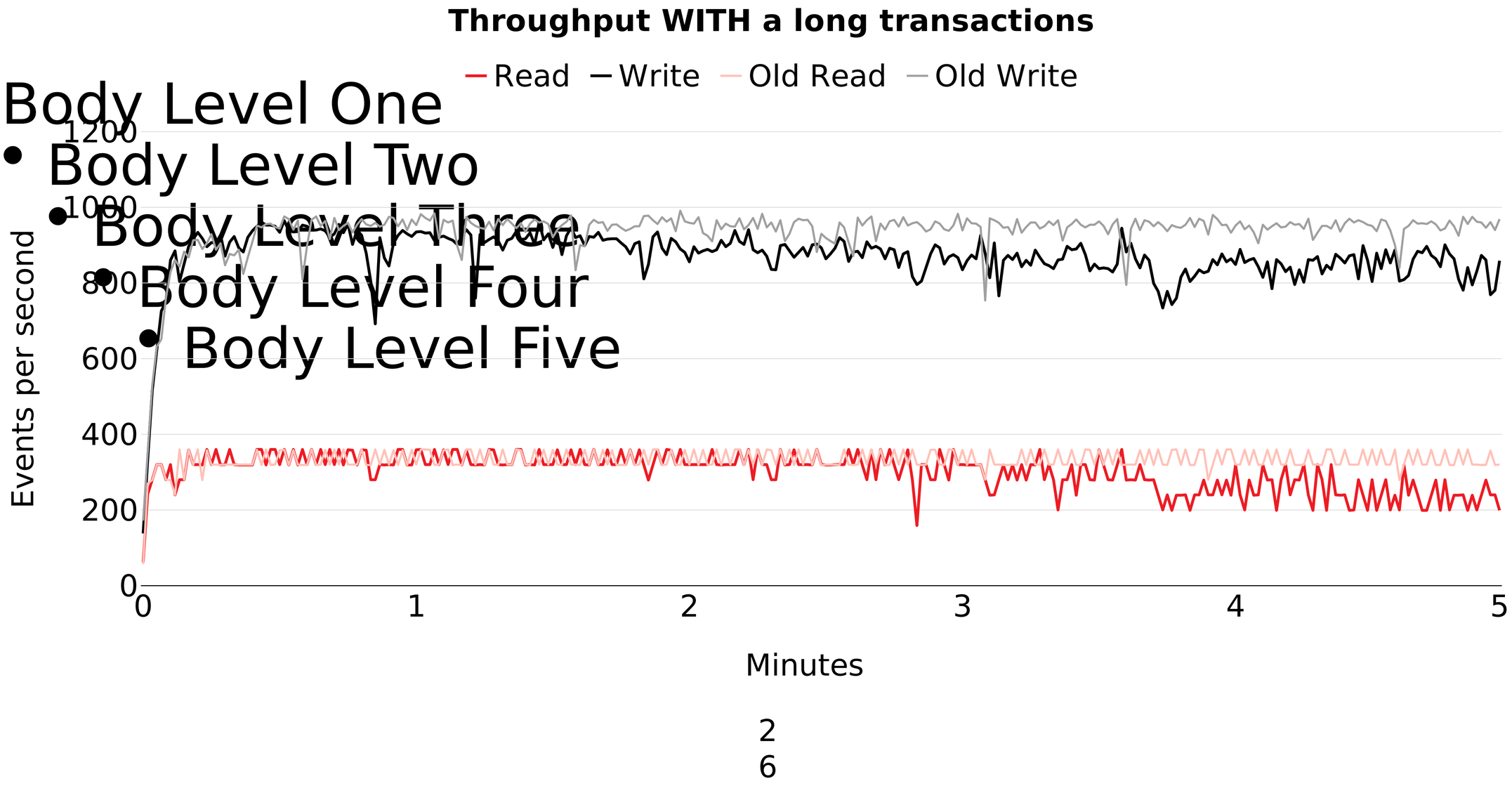
Title FOR UPDATE

- Body Level One
  - Body Level Two
    - Body Level Three
      - Body Level Four
        - Body Level Five



Title FOR UPDATE

- Body Level One
  - Body Level Two
    - Body Level Three
      - Body Level Four
        - Body Level Five



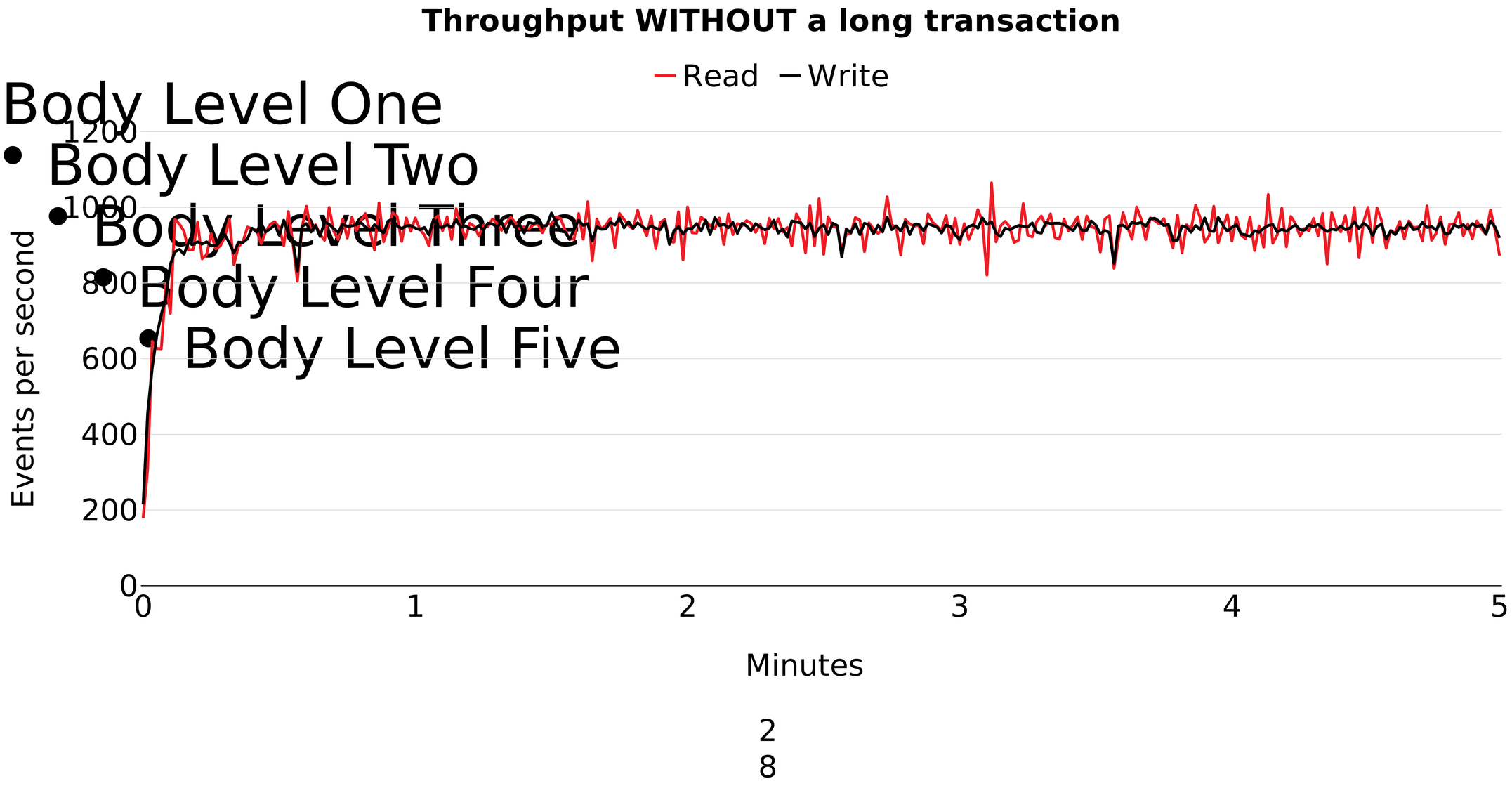
Zitsa SKIP LOCKED

- Body Level One
  - Body Level Two
    - Body Level Three
      - Body Level Four
        - Body Level Five

- SELECT ... FOR UPDATE **SKIP LOCKED**
- The reader concurrency problem is solved
- No order guarantee

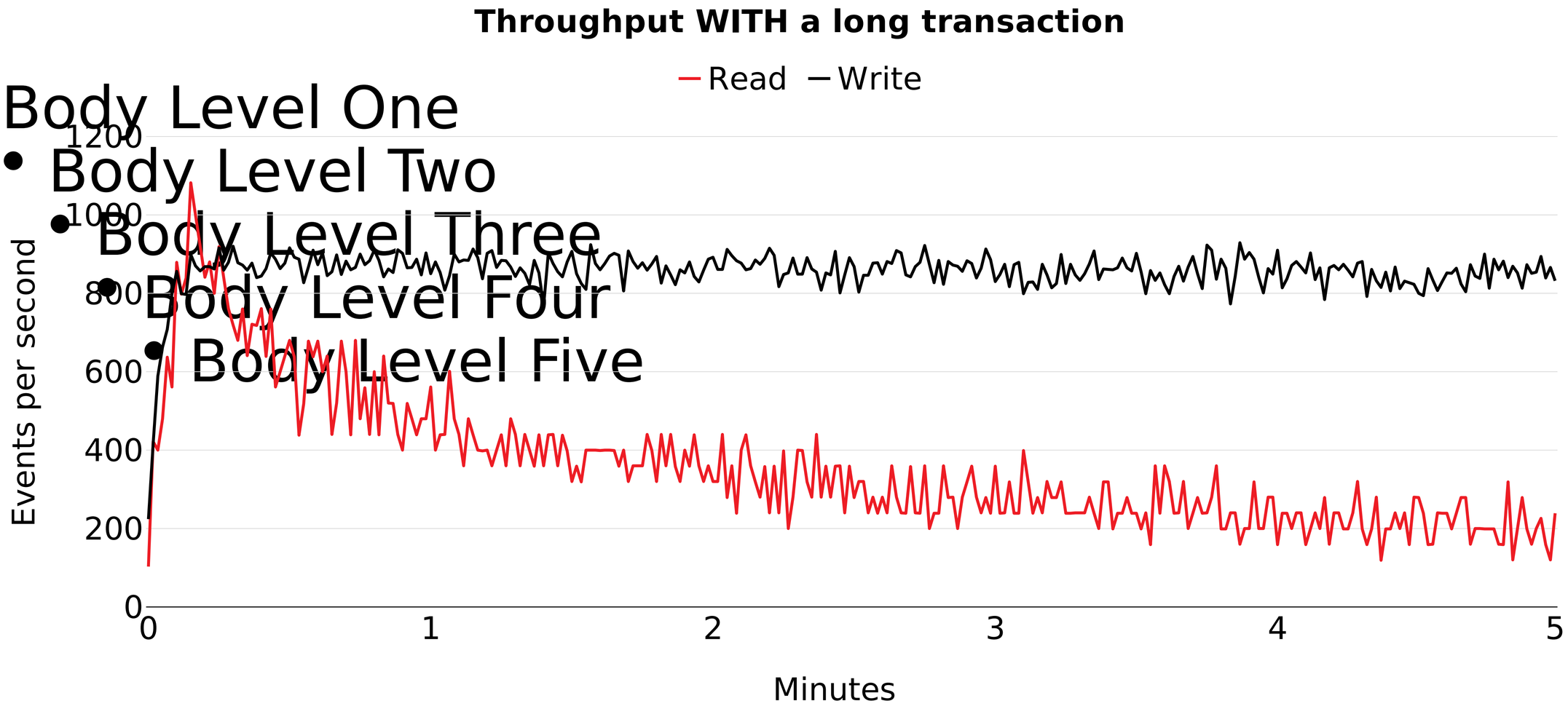
SKIP LOCKED

- Body Level One
  - Body Level Two
    - Body Level Three
      - Body Level Four
        - Body Level Five



SKIP LOCKED

- Body Level One
  - Body Level Two
    - Body Level Three
      - Body Level Four
        - Body Level Five



Bitwise WHERE id > :prev_id

- SELECT ... **WHERE id > :prev_id** ... FOR UPDATE SKIP LOCKED
- prev_id is the last processed ID
- Every 10th query will reset prev_id

- Body Level One
  - Body Level Two
    - Body Level Three
      - Body Level Four
        - Body Level Five

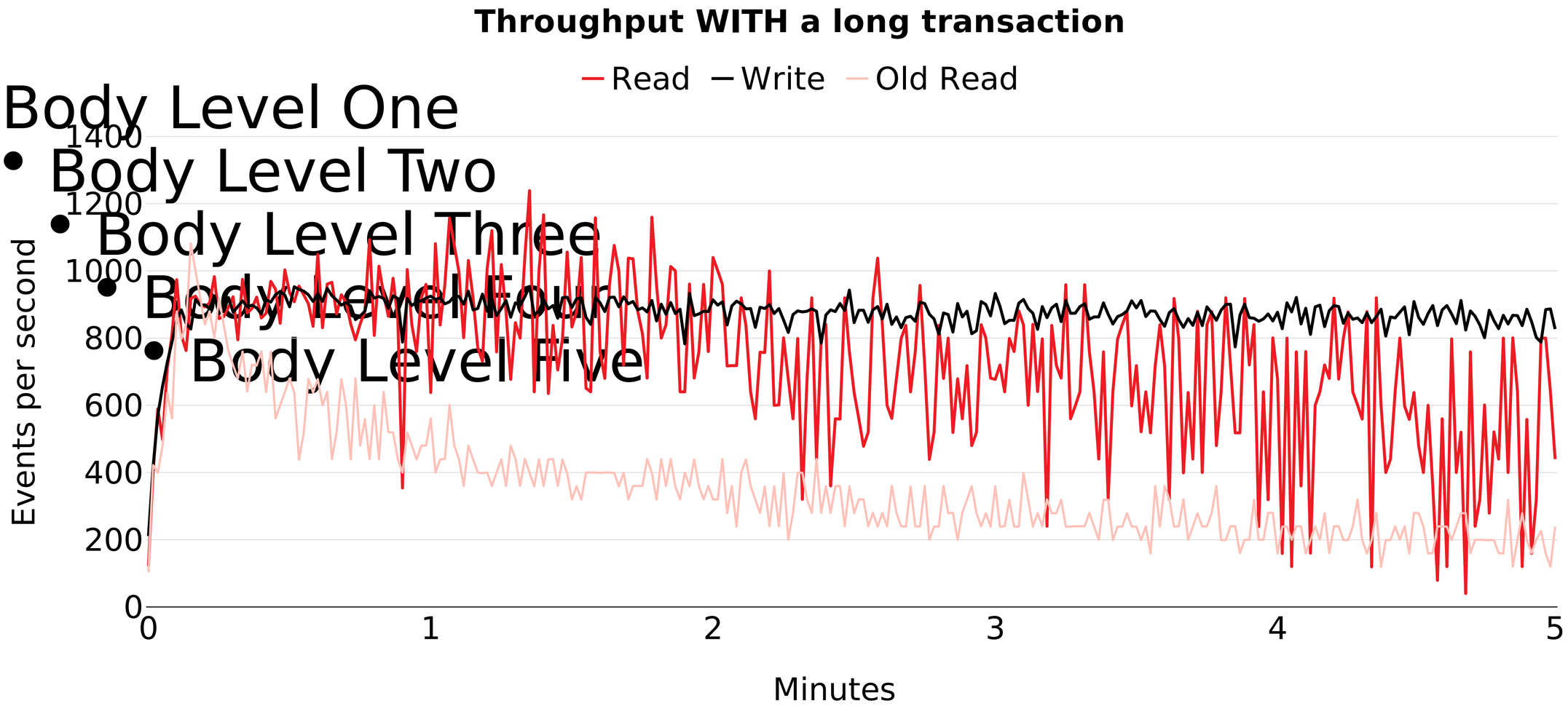
Bitwise WHERE id > :prev_id

- SELECT ... **WHERE id > :prev_id** ... FOR UPDATE SKIP LOCKED
- prev_id is the last processed ID
- Every 10th query will reset prev_id

- Body Level One
  - Body Level Two
    - Body Level Three
      - Body Level Four
        - Body Level Five

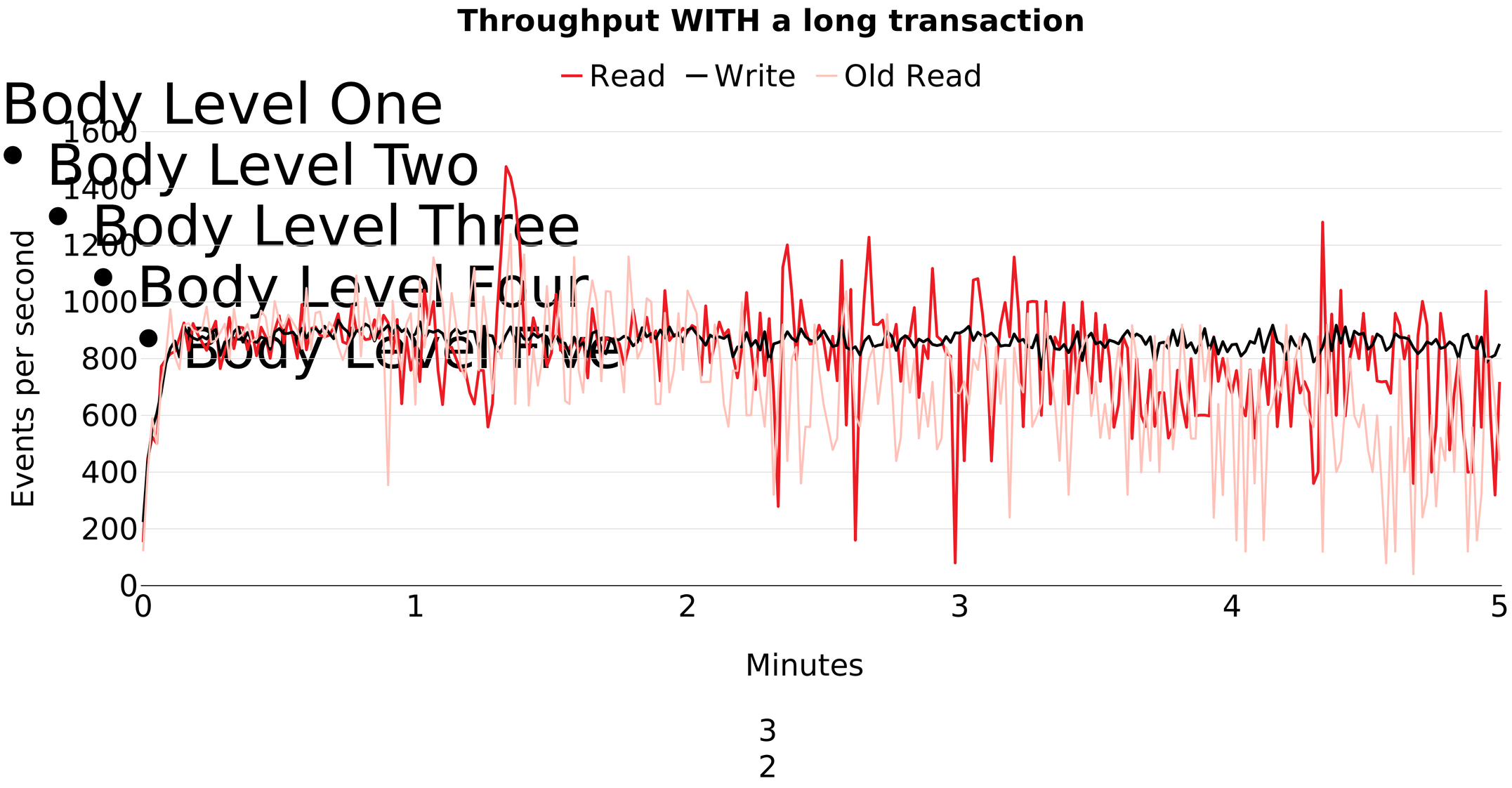
3itWHERE id > :prev_id

- Body Level One
  - Body Level Two
    - Body Level Three
      - Body Level Four
      - Body Level Five



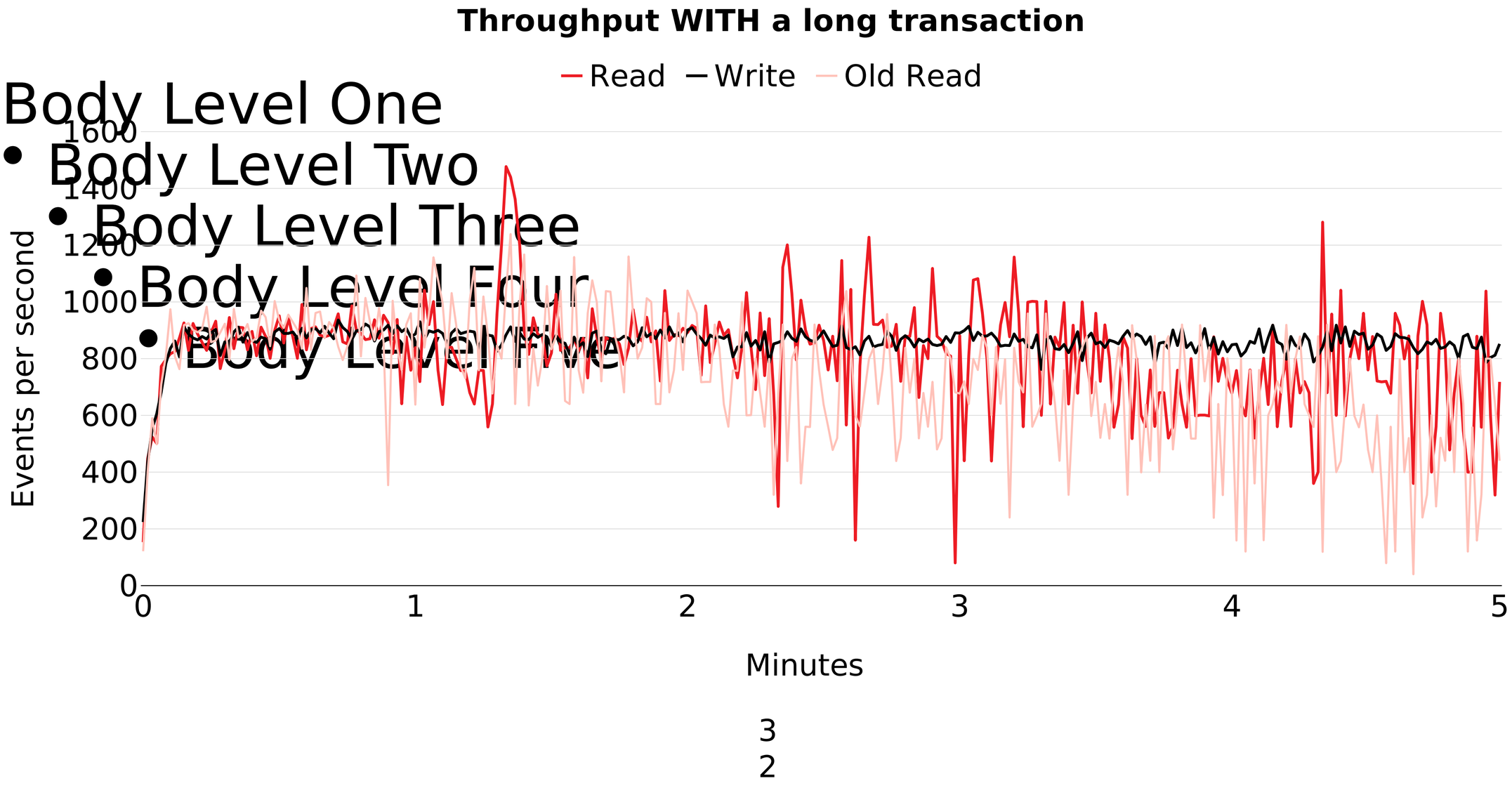
Title
SET LOCAL synchronous_commit TO OFF

- Body Level One
  - Body Level Two
    - Body Level Three
      - Body Level Four
        - Body Level Five



Title
SET LOCAL synchronous_commit TO OFF

- Body Level One
  - Body Level Two
    - Body Level Three
      - Body Level Four
        - Body Level Five



Title TRUNCATE

- Body Level One
  - Body Level Two
    - Body Level Three
      - Body Level Four
        - Body Level Five

- TRUNCATE is not MVCC safe
- Three queue_buffer tables instead of one
- Run TRUNCATE every 10 seconds

Title TRUNCATE

- Body Level One
  - Body Level Two
    - Body Level Three
      - Body Level Four
        - Body Level Five

- TRUNCATE is not MVCC safe
- Three queue_buffer tables instead of one
- Run TRUNCATE every 10 seconds

Title TRUNCATE

Writers will insert a message, if they can take a shared lock. N = 1, 2, 3

- Body Level One
  - Body Level Two
    - Body Level Three
      - Body Level Four
        - Body Level Five

```
BEGIN;
SELECT pg_try_advisory_xact_lock_shared(N);
INSERT INTO queue_buffer_N (payload) values (?);
COMMIT;
```

Title TRUNCATE

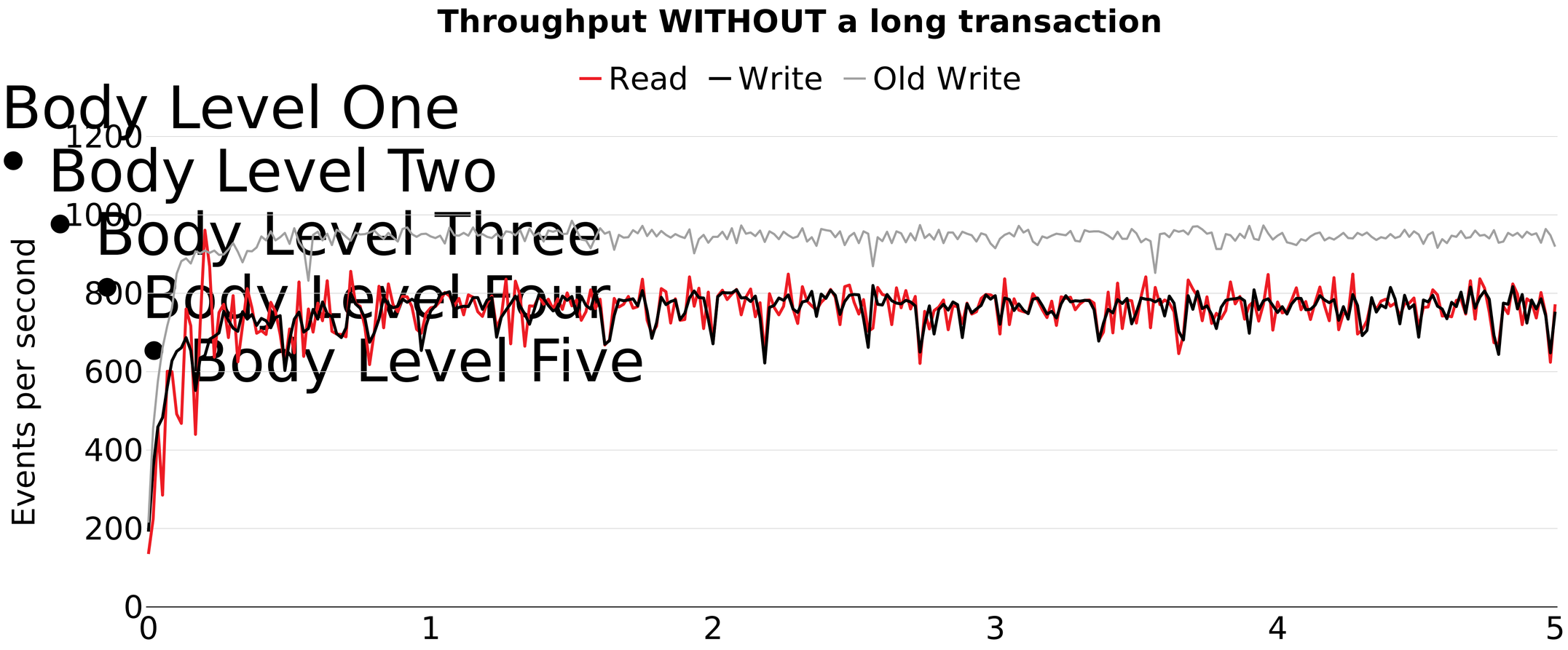
The TRUNCATE background process. N = 1, 2, 3

- Body Level One
  - Body Level Two
    - Body Level Three
      - Body Level Four
        - Body Level Five

```
BEGIN;
SET LOCAL lock_timeout = '30s'; SET LOCAL statement_timeout = '30s';
SELECT pg_advisory_xact_lock(N);
LOCK TABLE queue_buffer_N IN ACCESS EXCLUSIVE MODE;
CREATE TEMPORARY TABLE queue_buffer_N_copy ON COMMIT DROP AS SELECT * FROM queue_buffer_N
WITH DATA;
TRUNCATE queue_buffer_N;
INSERT INTO queue_buffer_N SELECT * FROM queue_buffer_N_copy;
COMMIT;
```

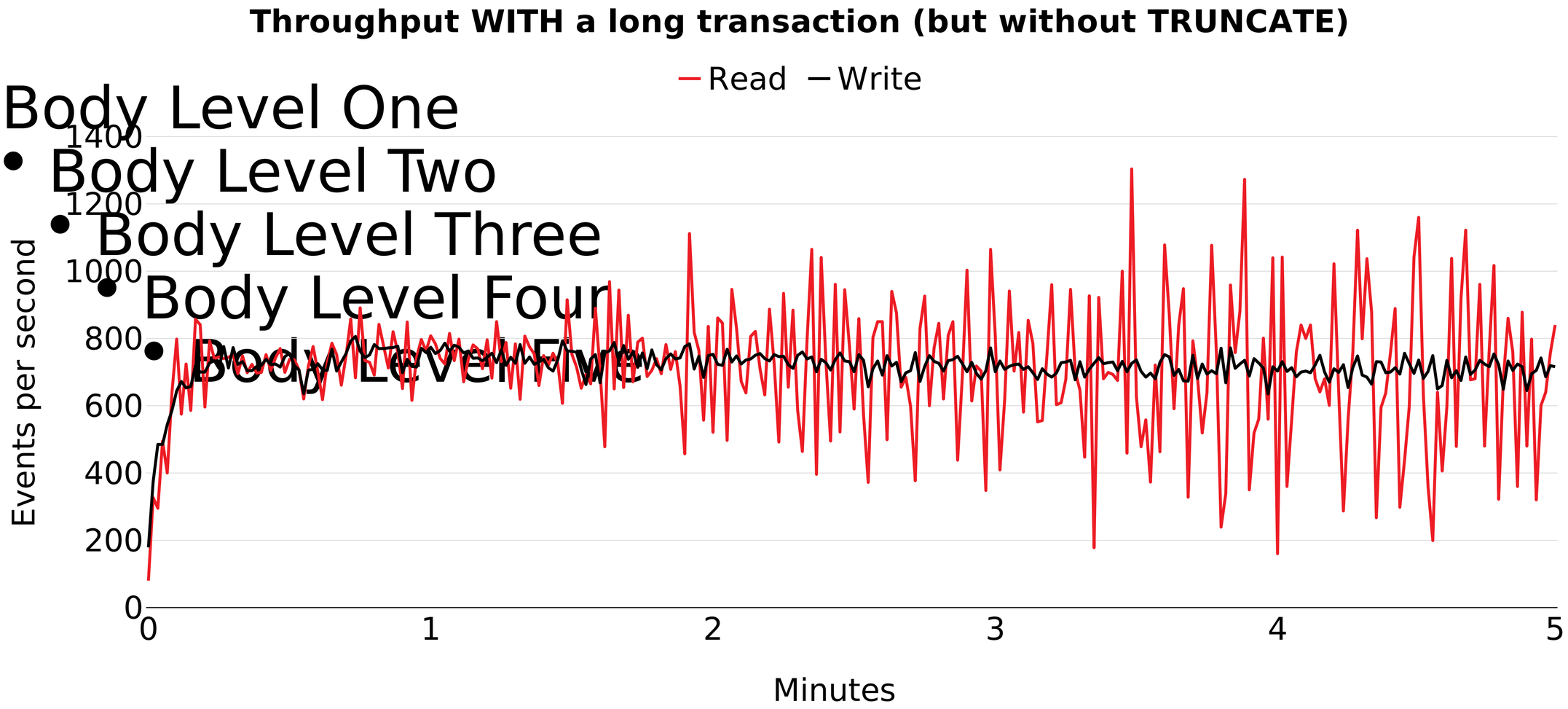
Title TRUNCATE

- Body Level One
  - Body Level Two
    - Body Level Three
      - Body Level Four
        - Body Level Five



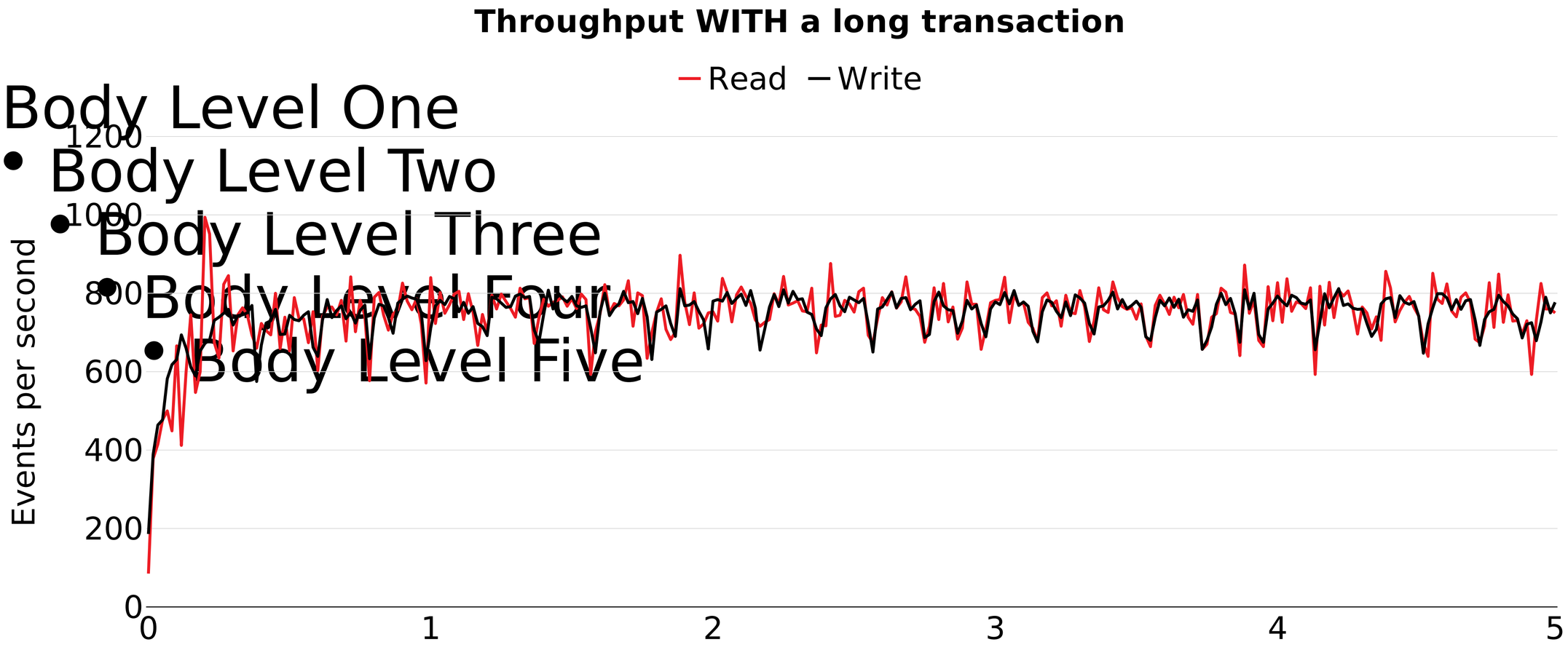
Title TRUNCATE

- Body Level One
  - Body Level Two
    - Body Level Three
      - Body Level Four
        - Body Level Five



Title TRUNCATE

- Body Level One
  - Body Level Two
    - Body Level Three
      - Body Level Four
        - Body Level Five



05

Summary

Playground

Toloka / pg-queue-playground

Watch 4 | Fork 0 | Starred 2

Code | Issues | Pull requests | Actions | Projects | Wiki | Security | Insights | Settings

main | 1 branch | 0 tags | Go to file | Add file | Code

Igor Loban Initial commit		2ecc4a8 2 hours ago	1 commit
gradle/wrapper	Initial commit		2 hours ago
pg-configs	Initial commit		2 hours ago
src	Initial commit		2 hours ago
.gitattributes	Initial commit		2 hours ago
.gitignore	Initial commit		2 hours ago

About

Playground for transactional queues in PostgreSQL

- Readme
- View license
- 2 stars
- 4 watching
- 0 forks

Summary

- We found out when transactional queues in DBs are needed
- Transactional Outbox pattern
- PgQ and Eventuate
- Performance tricks
- TRUNCATE against long transactions

Summary

- We found out when transactional queues in DBs are needed
- Transactional Outbox pattern
- PgQ and Eventuate
- Performance tricks
- TRUNCATE against long transactions

Summary

- We found out when transactional queues in DBs are needed
- Transactional Outbox pattern
- PgQ and Eventuate
- Performance tricks
- TRUNCATE against long transactions

- Body Level One
  - Body Level Two
    - Body Level Three
      - Body Level Four
        - Body Level Five

Kindly remind about **metrics**, **alerts**, and **load testing** for your solutions

Titles

- Body Level One
  - Body Level Two
    - Body Level Three
      - Body Level Four
        - Body Level Five

Design patterns

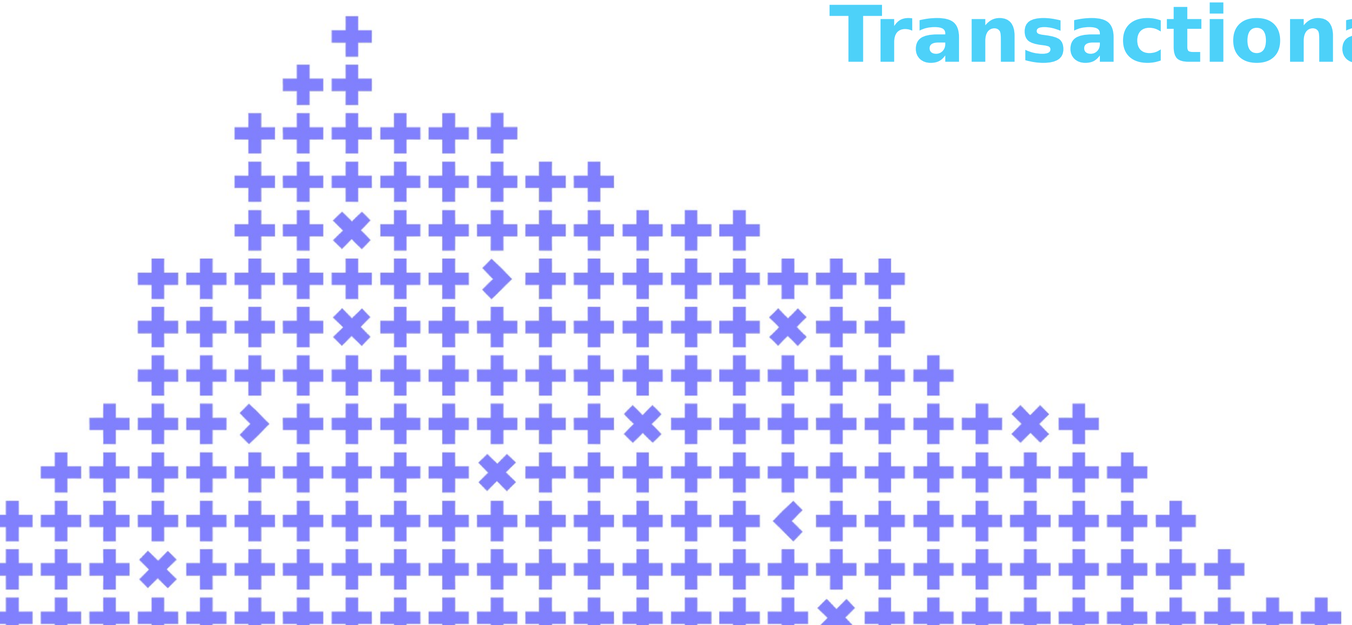
- https://microservices.io/patterns/data/transactional-outbox.html
- https://microservices.io/patterns/data/transaction-log-tailing.html
- https://microservices.io/patterns/data/polling-publisher.html
- https://microservices.io/patterns/data/saga.html

PgQ and Eventuate

- https://www.pgcon.org/2009/schedule/attachments/91_pgq.pdf
- https://habr.com/ru/post/483014/
- https://github.com/eventuate-tram

Leave your feedback!

You can rate the talk and give feedback on what you've liked or what could be improved



Transactional Queues in PostgreSQL

Igor Loban

High Load Armenia

Co-organizer

Yandex